Jenkins - Table of Contents

Jenkins Installation

- Launch an EC2 instance with Amazon Linux 2 with below userdata

```
#!/bin/bash
sudo yum update -y
sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-
ci.org/redhat/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat/jenkins.io.key
sudo yum install java-1.8.0 -y
sudo yum install jenkins -y
sudo yum install tree -y
sudo service jenkins start
sudo systemctl enable jenkins
```

This package installation will:

- Setup Jenkins as a daemon launched on start. See `/etc/init.d/jenkins` for more details.
- Create a `jenkins` user to run this service.
- Direct console log output to the file `/var/log/jenkins/jenkins.log`.
- Check this file if you are troubleshooting Jenkins.
- Populate `/etc/default/jenkins` with configuration parameters for the launch, e.g JENKINS_HOME
- Set Jenkins to listen on port 8080. Access this port with your browser to start configuration.
- Login to EC2 Jenkins Server using ssh.

```
netstat -nltp
sudo service jenkins status
sudo service jenkins stop
sudo service jenkins restart
```

- Check Jenkins Port Information

```
sudo cat /etc/sysconfig/jenkins | grep -i port
ps -elf | grep 8080
```

- Lets look at the jenkins log file

```
sudo cat /var/log/jenkins/jenkins.log
```

- Access the Jenkins UI

```
http://public-ip:8080
```

- Admin Password is written to a file

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

- Select `Install Suggested Plugins` only

- It will ask for creating for first admin user, enter details as required.

- A linux user with name `jenkins` is created while Jenkins Installation, add `jenkins` user in linux to `sudoers` group

```
cat /etc/passwd | grep -i 'jenkins'
sudo usermod -a -G wheel jenkins
id jenkins
```

- Current home directory for jenkins is **/var/lib/jenkins**
- To view the Jenkins Systems Information, navigate to `Manage Jenkins` > `System Information`

Setup JDK for Jenkins

- Install OpenJDK 8 JDK
- To install OpenJDK 8 JDK using yum, run this command:

```
sudo yum install java-1.8.0-openjdk-devel
```

- Use below find command to search for files with name "jdk"

```
sudo find / -name "*jdk*"
```

Provide the path under provide the path under : Go to `Jenkins Dashboard -> Manage Jenkins -> Global Tool Configuration > JDK` > Give a Name > Give appropriate path to JDK e.g `/usr/lib/jvm/java-1.8.0-openjdk`

Installing the Git plugin

- We need to install the Git client on to our Jenkins server

```
sudo yum install git -y
git --version
```

- Check if Git Plugin is installed under: `Manage Jenkins > Manage Plugins > Installed Tab > Filter "Git Plugin"` , if not install it from `available` tab.
- This will prompt Jenkins to download the plugin, install it, and restart Jenkins to make it available for use.

Create a Jenkins Freestyle Project

- Click on **New Item** then enter an item name, select **Freestyle project**.
- Enter some build commands i.e bash commands `printenv` to be executed in the freestyle project.

Build with Parameters

- Sometimes, it is useful/necessary to have your builds take several "parameters."
- This can to run a Pipeline Job as per SDLC Environment or any other value to be passed on Job Runtime.
- Under a specific jenkins project, select `Configure` option, select the checkbox `This project is parameterized` and `Add Parameter`
- For testing the value of the runtime parameter, keep `Source Code Management` as `None`
- The parameters are available as `environment variables`. So a shell $PARAM_VAR,can be used to access these values.

Integrate Jenkins with Github Repo

- Select the GitHub project checkbox and set the Project URL to point to your GitHub Repository. `https://github.com/YourUserName/`

- Under Source Code Management Section: Provide the Github Repository URL where Source Code is present, keep the branch as master.
- Go to Jenkins Project -> Configure -> Under Build Environment Build Step > Select "Execute Shell Script" from dropdown > `write shell commands`

> Execute a shell script stored in Github repo by providing path.

- Click on **Build Now** to Build this Project

Jenkins jobs and workpace information

The Jenkins home directory structure Colons can be used to align columns.

Directory	Description
jobs	It contains configuration details about the build jobs that Jenkins manages, as well as the artifacts and data resulting from these builds.
workspace	It is where Jenkins builds your project: it contains the source code Jenkins checks out, plus any files generated by the build itself.This workspace is reused for each successive build, there is only ever one workspace directory per project, and the disk space it requires tends to be relatively stable.

Jenkins Environment Variables:

- To view all the environment variables simply append `env-vars.html` to your Jenkins Server's URL.
- Create a simple free style job to display the value of the environment variables that are set for a Jenkins Job:
- Under Build Section > Add build step > Execute shell , add below commands:

```
echo "BUILD_NUMBER" :: $BUILD_NUMBER
echo "BUILD_ID" :: $BUILD_ID
echo "BUILD_DISPLAY_NAME" :: $BUILD_DISPLAY_NAME
echo "JOB_NAME" :: $JOB_NAME
echo "EXECUTOR_NUMBER" :: $EXECUTOR_NUMBER
echo "NODE_NAME" :: $NODE_NAME
echo "NODE_LABELS" :: $NODE_LABELS
echo "WORKSPACE" :: $WORKSPACE
echo "JENKINS_HOME" :: $JENKINS_HOME
echo "JENKINS_URL" :: $JENKINS_URL
echo "BUILD_URL" ::$BUILD_URL
echo "JOB_URL" :: $JOB_URL
echo "Below output is all the environment variable in Jenkins"
printenv
```

- The `printenv` command prints all the Jenkins Environment Variables set for that specific Build.

Generate SSH keys and add to Github:

- Generate ssh keys and add to `Github Account` **OR** `Specific Github Repo`

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

- To configure ssh keys for Github Account follow below steps:
    - This creates a new ssh key, using the provided email as a label.
    - Add the public ssh key to github account under `Settings > SSH and GPG Keys > New SSH key > Add SSH Key`, confirm your password.
- To configure ssh keys for private Github repository follow below steps:
    - Go to your private Github repo -> Settings -> Deploy keys.
    - This creates a new ssh key, using the provided email as a label.
    - Add your ssh key inside the key text input folder, if you want write access to the repo click on the Allow write access.
- Jenkins configuration to access private repo:
    - Navigate to `Jenkins dashboard -> Credentials -> System -> Global credentials -> Add credentials.`
    - Give username as Jenkins or any value, Add the Private key -> Enter directly and copy paste the same ssh keys here.
- While setting up a Job in Jenkins, add the credentials created above to the credentials section in `Source Code Management` under the Repository URL.

Setup maven

- Go to `Jenkins Dashboard` -> `Manage Jenkins` -> `Global Tool Configuration` > `Maven` > Give a Name `Maven_Local` > Check `Install Automatically` > Install from Apache (specify a version) > `Save`
- You can give a logical name to identify the correct version while configuring a build job

Jenkins with Maven Build

- Click on **New Item** then enter an item name, select **Freestyle project**.
- Under Source Code Management tab, select Git and then set the Repository URL to point to your GitHub Repository. `https://github.com/YourUserName/repo-name.git`
- Under Build Environment Build Step > Select `Invoke top-level Maven targets` from dropdown > select the Maven Version that we just created, specify `clean install`.
- Under Advanced tab, specifiy the pom.xml file relative path location from git repository.
- Click on `Save`

> it will run command `mvn clean install -f pom.xml`

- Click OK and Build a Job and you will see that a war file is created.

Maven build phases

- Maven itself requires Java installed on your machine.
- You can verify if Maven is installed on your machine by running "mvn -v" in your command line/terminal.
- Maven is based on the `Project Object Model (POM)` configuration, which is stored in the XML file called the same – `pom.xml`. It is a structured format that describes the project, it's dependencies, plugins, and goals. `pom.xml` file in your project directory
- **Validate** : Validate Project is correct & all necessary information is available.
- **Compile** : Compile the Source Code
- **Test** : Test the Compiled Source Code using suitable unit Testing Framework (like JUnit)
- **package** : Take the compiled code and package it.
- **Install** : Install package in Local Repo, for use as a dependency in other project locally.
- **Deploy** : Copy the final package to the remote repository for sharing with other developers.
- The above are always are sequential, if you specify `install`, all the phases before `install` are checked.

Artifacts Archive

- Go to `Jenkins dashboard` -> `Jenkins project or build job` -> `Post-build Actions` -> `Add post-build action` -> `Archive the artifacts`:
- Enter details for options in `Archive the artifacts` section:
  - For `Files to archive` enter the Path of the `.war` file like : `java-tomcat-sample/target/*.war`
- `Save` the changes and `Build Now`.
- Check the directories as below to validate above information:

```
ls /var/lib/jenkins/jobs
ls /var/lib/jenkins/jobs/<JOB_NAME>
ls /var/lib/jenkins/jobs/<JOB_NAME>/builds/<BUILD_NUMBER>
ls /var/lib/jenkins/workspace/<JOB_NAME>
```

- If you check the directory structure, there will be archive directory present under the subsequent build number for which the job is executed with Post build action as `Archive the artifacts`

Managing access control and authorization

- Managing access control and authorization

  - Go to `Manage Jenkins` > `Configure Global Security` > `Enable security`.
- On the Jenkins dashboard, click on Manage Jenkins > Manage Users.
  - We can edit user details on the same page. This is a subset of users, which also contains auto-created users.

Maintaining roles and project-based security

- Lets configure some users in Jenkins, create a read only user `readonlyuser`
  - Select `Manage Jenkins` > `Manage Users` > `Create a user` For authorization, we can define Matrix-based security on the Configure Global Security page.

1. Add group or user and configure security based on different sections such as Credentials, Slave, Job, and so on.
2. Click on Save.

- Try to access the Jenkins dashboard with a newly added user who has no rights, and we will find the authorization error.

Role-Based-Authorization Strategy

- Add plugin from available tab in Plugins Manager.
- Go to `Manage Jenkins` > `Configure Global Security` > Select the `Role-based Authorization Strategy`
- Create a role with `Manage and Assign Roles` > `Manage Roles` > Create a `ReadOnlyRole` > select Read Permissions.
- Assign this role to another user.